Functional Testing of Core Requirements

David Flater

2007-03-21 10:53

1 Introduction

This is the documentation for the Votetest distribution. Its intended audience is test labs accredited by the Election Assistance Commission to perform federal voting system certification testing. The reader is assumed to have knowledge of the following:

- The Voluntary Voting System Guidelines (VVSG) [1];
- Data modelling;
- Structured Query Language (SQL) [2];
- The use and conventions of Unix operating systems.

The Votetest distribution contains the following components for functional testing of core requirements to the Voluntary Voting System Guidelines (VVSG).

- The data model (Section 2) documents the world view inherent in the schema and test suite. It is general enough to support arbitrary combinations of all of the voting variations defined in the VVSG.
- The schema (Section 3) is an SQL realization of the data model and the logic model of the VVSG, which specifies the results that voting systems are required to report.
- The test suite (Section 4) contains test cases built upon the schema and infrastructure needed to execute them.

Since there is no standard interface to voting systems, it is anticipated that test labs will use the test cases included in this distribution as the input from which to generate vendor-specific test cases for use in federal certification testing. The schema's realization of the VVSG logic model serves as a test oracle to determine the expected results for each test case.

The schema for functional testing of core requirements is designed only for that purpose. It does not respond to the security, privacy, accessibility, or usability requirements of the VVSG and does not compete with available voting systems or standards.

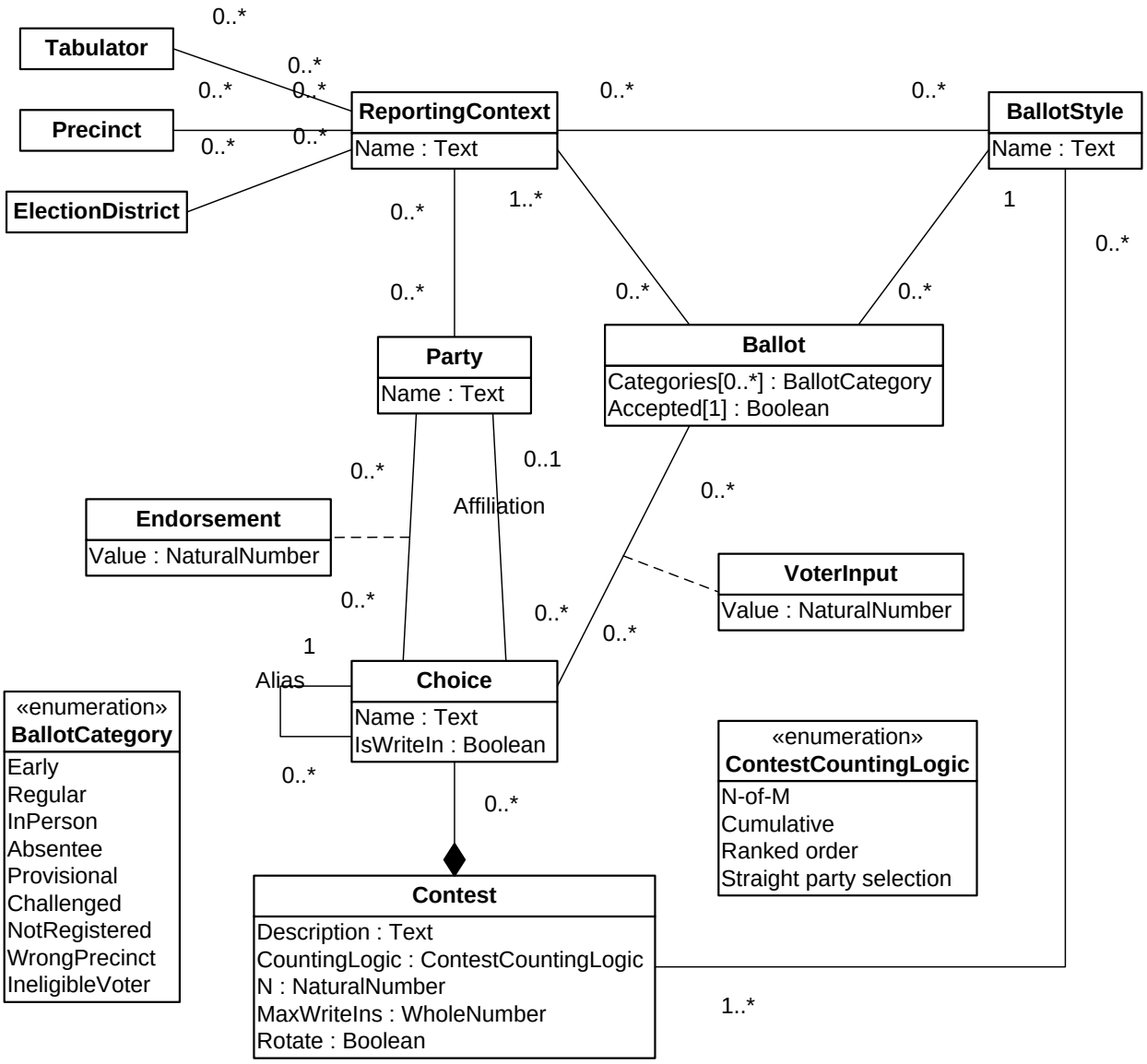

Figure 1: Vote data model for functional testing of core requirements

2 Data Model

The data model used for functional testing of voting system core requirements is described in Figure 1 by a class diagram conforming to version 1.5 and subsequent versions of Unified Modeling Language [3]. Following sections explain the diagram.

2.1 Assumptions

All entities in this data model are implicitly scoped by an election. It is assumed that different elections are stored in different databases, and any reuse of ballot styles and other data from one election to another is accomplished by copying over the relevant data.

This data model is constructed from an integrated, top-level viewpoint. In practice, different portions of the system will deal with only a portion of the data at any given time. It is expected that data will be projected and extracted from the integrated schema as needed to support these limited viewpoints in testing.

The results of tabulation and reporting are derived from the content of the data model, but those results are themselves outside the scope of the model.

2.2 POD (Plain Old Data) types

BallotCategory (enum) Arbitrary tag that may be applied to Ballots. Categories are jurisdiction-defined but are likely to include several classes of provisional.

Boolean Normal true/false data type.

ContestCountingLogic (enum) N-of-M, Cumulative, Ranked order, or Straight party selection. (1-of-M is a special case of N-of-M.) The tabulation logic for a straight party selection Contest is implicitly 1-of-M, but with side-effects for other contests.

NaturalNumber Integer greater than zero.

Text Normal character string.

WholeNumber Integer greater than or equal to zero.

2.3 Classes

2.3.1 Ballot

The undefined primitive in all elections. The contests that appear on a particular ballot are defined by its BallotStyle. The applicable ReportingContexts include all those specified for its BallotStyle, but additional contexts may be specified for the individual ballot.

Attributes of Ballot:

Categories Arbitrary, jurisdiction-defined tags applied to the Ballot.

Accepted True if the ballot should be counted, false if not (e.g., for a provisional ballot that was not accepted).

2.3.2 BallotStyle

Set of Contests and ReportingContexts that is inherited by all Ballots of that style. Depending on the type of election and local practices, a jurisdiction would define a separate BallotStyle for each precinct, each split within a precinct, and/or for each political party.

Attributes of BallotStyle:

Name Human-readable identifier.

2.3.3 Choice

One of the things you can vote on in a Contest, such as a candidate, a political party, or yes or no. Choice is scoped by Contest, so even if the same person runs as a candidate in two or more Contests, those separate candidacies are represented by separate Choices. Choices do not map 1:1 with ballot positions—a Choice uniquely identifies a candidate, while a given ballot position might just be a generic write-in slot.

Attributes of Choice:

Name Human-readable identifier. (In a real system, Choices could have complex descriptive data associated with them that must be displayed to the user somehow, but for testing purposes a single field suffices.)

IsWriteIn True if the Choice is a write-in candidate, false if not.

2.3.4 Contest

Subdivision of a Ballot corresponding to a single question being put before the voters, consisting of header text, a discrete set of choices, and possibly write-in opportunities. It is possible for a Contest to have zero Choices, e.g., if there are no registered candidates but write-ins are being accepted. Choices corresponding to the candidates written in would be added later.

Attributes of Contest:

Description Human-readable header text.

CountingLogic Identifies the tabulation method used for the contest.

N The maximum number of choices that a voter may make in an N-of-M contest, the maximum number of votes that the voter may allocate in a cumulative contest, or the maximum number of candidates that the voter may rank in a ranked order contest, without overvoting. The value of M, for N-of-M voting, is simply the number of Choices associated with the Contest and is not explicitly modelled. N may exceed M if the number of open seats exceeds the number of candidates.

MaxWriteIns The number of ballot positions allocated for write-ins; the maximum number of candidates that the voter may write in. Any value between zero and N is possible. Zero would mean that write-ins are not allowed; N would mean that write-ins are allowed; a number in between would mean that write-ins must be approved and the number of approved write-in candidates is less than N.

Rotate True if the ordering of Choices within the Contest should be rotated, false if not.

2.3.5 ElectionDistrict

Surrogate for real-world entity that may have associated ReportingContexts.

2.3.6 Party

Surrogate for real-world political party.

Attributes of Party:

Name Unique human-readable identifier.

2.3.7 Precinct

Surrogate for real-world entity that may have associated ReportingContexts.

2.3.8 ReportingContext

Particular scope within which the system must be capable of generating reports. E.g., to support reporting at the precinct level, there must be a ReportingContext for each precinct. The association between ReportingContexts and individual tabulators, precincts, election districts, jurisdictions, political parties, ballot categories, or other arbitrary scopes of reporting is jurisdiction-defined and jurisdiction-managed, mostly using BallotStyles. The ways in which ReportingContexts overlap or include one another is entirely determined by the assignment of multiple ReportingContexts to BallotStyles and Ballots.

Attributes of ReportingContext:

Name Human-readable identifier.

2.3.9 Tabulator

Surrogate for real-world entity that may have associated ReportingContexts.

2.4 Named associations

2.4.1 Affiliation

Identifies the Party to which a candidate claims allegiance. Does not necessarily have anything to do with Endorsements.

2.4.2 Alias

Identifies an alternative Choice that for tabulation purposes is considered equivalent to a particular canonical Choice. Aliases will normally be variant spellings of a candidate's name that appeared in write-in positions.

2.4.3 Endorsement

Identifies a voter response that would be implied by a straight party vote for the endorsing Party. Does not necessarily have anything to do with Affiliation.

Attributes of Endorsement:

Value Analogous to VoterInput Value, this is the vote recommended by the endorser.

2.4.4 VoterInput

The response that a particular Ballot provides for a particular Choice.

Attributes of VoterInput:

Value The response of the voter in some ballot position. The absence of a response is equivalent to a Value of 0.

2.5 Constraints

I. For N-of-M contests, the Value attribute of VoterInput or Endorsement must be 1. For cumulative and ranked order contests, $1 \leq$ Value $\leq$ N.

II. $0 \leq$ MaxWriteIns $\leq$ N.

III. In Contests with CountingLogic = Straight party selection, N = 1 and MaxWriteIns = 0.

IV. Every Ballot must be associated with at least one ReportingContext either directly or through its BallotStyle. (Otherwise the ballot would never be reported.)

V. A Ballot cannot have a VoterInput for a Choice in a Contest that does not appear in its BallotStyle.

VI. A given BallotStyle may contain at most one Contest with CountingLogic = Straight party selection.

VII. A Contest with CountingLogic = Straight party selection cannot be straight-party-votable (i.e., there can be no Endorsements referring to its Choices).

VIII. In Contests with CountingLogic = Straight party selection, the Names of the Choices must match the Names of Parties.

IX. Party names must be unique.

X. A Ballot may not simultaneously have VoterInput for a Choice and an Alias of that Choice. (The handling of double votes for a given candidate resulting from write-in reconciliation is deliberately unspecified in the VVSG, so for testing purposes it is considered an error.)

XI. A Ballot may not simultaneously have VoterInput in a straight-party-votable Contest and a straight party vote that implies votes in that same Contest. (Resolution of scratch votes is deliberately unspecified in the VVSG, so for testing purposes they are considered to be errors.)

XII. The Choice that an Alias cites as canonical cannot be aliased. (Corollary: There can be no cycles or self-referential Aliases.)

XIII. The Choice that an Alias cites as canonical must be in the same Contest.

XIV. The Choice referenced by an Endorsement must be canonical (it cannot be an Alias).

XV. A Ballot cannot have VoterInput for more write-in Choices in a given Contest than is allowed by the MaxWriteIns attribute of the Contest.

2.6 Usage for all standard voting variations

2.6.1 In-person voting

No special requirements.

2.6.2 Absentee voting

Absentee voting is implemented in several different ways in practice, and it can be implemented in several different ways using this model.

1. Absentee ballots can be tagged with the Absentee category and otherwise mingled with other ballots.
2. A separate ReportingContext can be created for absentee ballots and applied to the individual absentee ballots.
3. A separate BallotStyle can be used for absentee ballots.

While the first option is the least invasive, absentee ballots are in practice sometimes processed as a separate precinct, which usually means both a separate ReportingContext and a separate BallotStyle.

2.6.3 Review-required ballots

Use Categories and Accepted attributes of Ballot as needed.

2.6.4 Write-ins

The number of write-ins permitted is an attribute of the Contest. If the write-in is new, a new Choice is created for it (with IsWriteIn = true). Votes are then associated with that Choice. Alias associations are created as applicable during write-in reconciliation.

2.6.5 Split precincts

Ballots are associated with the ReportingContexts pertaining to the applicable Precinct and ElectionDistrict. If different BallotStyles are used for each split, the associations can be made on the BallotStyles. Otherwise, each Ballot must be individually associated.

2.6.6 Straight party voting

A single contest is created with CountingLogic = Straight party selection and Choice Names being equal to the Names of the available Parties. In every other contest that is straight-party-votable, the straight party behaviors are configured by creating Endorsement associations between the Choices and the Parties.

2.6.7 Cross-party endorsement

See straight party voting. Create additional Endorsement associations as needed for multiply endorsed candidates.

2.6.8 Ballot rotation

Rotate is a Boolean attribute of Contest. The implementation of variable mapping between Choices and ballot positions is out of scope because ballot positions are abstracted out of the model. However, in paper-based systems, rotation may involve a proliferation of BallotStyles that would have to be added.

2.6.9 Primary elections

Create BallotStyles and ReportingContexts as needed to support the different political parties and unaffiliated voters. Nonpartisan contests appear in all BallotStyles while partisan contests only appear in those BallotStyles applicable to the relevant Party.

2.6.10 Closed primaries

Assignment of BallotStyles to voters is procedural and out of scope.

2.6.11 Open primaries

Assignment of BallotStyles to voters is procedural and out of scope.

2.6.12 Provisional / challenged ballots

Use Categories and Accepted attributes of Ballot as needed.

2.6.13 1-of-M voting

Set ContestCountingLogic = N-of-M and set N = 1.

2.6.14 N-of-M voting

Set ContestCountingLogic = N-of-M and set N appropriately.

2.6.15 Cumulative voting

Set ContestCountingLogic = Cumulative and set N appropriately.

2.6.16 Ranked order voting

Set ContestCountingLogic = Ranked order and set N appropriately. VoterInput Values specify the rankings as provided on each Ballot.

3 Schema

The schema for functional testing of core requirements is built in five layers.

1. Translation of the data model. This layer contains all of the tables and data. The other layers are comprised entirely of views.
2. Conveniences defined over the data model.
3. Adaptation layer. This layer translates the raw voter inputs per the data model into the effective voter inputs required by the logic model.
4. Integrity checks.
5. Translation of the logic model.

The schema was tested with PostgreSQL 8.2.3 [4] running on a GNU/Linux operating system. It uses extensions to the SQL standard [2] that might not function as intended with other databases.

Specific software is identified in this paper to foster understanding. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the software identified is necessarily the best available for the purpose.

3.1 Translation of data model

The following transforms are used to render the UML model as SQL.

1. At the most basic level, a table represents a class, the columns of the table represent the attributes of that class, and the rows of the table represent the instances of that class.
2. Object identity (*haecceity*) is implemented either by using an existing identifier as primary key or by using a synthetic identifier of integer type, as convenient.
3. Associations to at most 1 instance of another class are implemented using foreign keys within the relevant table with a not-null constraint if the minimum multiplicity is 1. Associations of higher multiplicity are reified as separate tables.
4. Attributes of multiplicity greater than 1 are treated as associations and reified as separate tables.

5. Enums are implemented using the names of the enum values as identifiers. Integrity is maintained by creating a table containing the enum values and making attributes of that enum type into foreign keys on that table.

The classes Tabulator, Precinct and ElectionDistrict are not represented. They were modelled only to clarify how the more general concept ReportingContext relates to the real world and are not needed by the test suite.

```
-- enum
create table BallotCategory (
  Name  Text  primary key
);
insert into BallotCategory values
  ('Early'), ('Regular'), ('InPerson'), ('Absentee'), ('Provisional'),
  ('Challenged'), ('NotRegistered'), ('WrongPrecinct'), ('IneligibleVoter');


-- enum
create table ContestCountingLogic (
  Name  Text  primary key
);
insert into ContestCountingLogic values
  ('N-of-M'), ('Cumulative'), ('Ranked order'), ('Straight party selection');


-- class
create table ReportingContext (
  Name  Text  primary key
);


-- class
create table Party (
  Name  Text  primary key
);


-- class
create table Contest (
  ContestId      Integer  primary key,
  Description    Text     not null,
  CountingLogic  Text     not null references ContestCountingLogic,
  N              Integer  not null check (N > 0),
  MaxWriteIns    Integer  not null check (MaxWriteIns between 0 and N),
  Rotate         Boolean  not null,

  -- Straight party selections must be 1-of-M with no write-ins.
  check (CountingLogic <> 'Straight party selection' or
          (N = 1 and MaxWriteIns = 0))
```

```
);


-- class
create table Choice (
  ChoiceId     Integer  primary key,
  ContestId    Integer  not null references Contest,
  Name         Text     not null,
  Affiliation  Text     references Party,    -- named association
  IsWriteIn    Boolean  not null
);


-- class
create table BallotStyle (
  StyleId  Integer  primary key,
  Name     Text     not null
);


-- class
create table Ballot (
  BallotId  Integer  primary key,
  StyleId   Integer  not null references BallotStyle,
  Accepted  Boolean  not null
);


-- attribute Ballot::Categories
create table BallotCategoryAssociation (
  BallotId  Integer  references Ballot,
  Category  Text     references BallotCategory,
  primary key (BallotId, Category)
);


-- association class
create table VoterInput (
  BallotId  Integer  references Ballot,
  ChoiceId  Integer  references Choice,
  Value     Integer  not null check (Value > 0),
  primary key (BallotId, ChoiceId)
);


-- association class
create table Endorsement (
  Party     Text     references Party,
```

```
  ChoiceId  Integer  references Choice,
  Value     Integer  not null check (Value > 0),
  primary key (Party, ChoiceId)
);


-- named association
create table Alias (
  AliasId   Integer  primary key references Choice,  -- The unwanted alias
  ChoiceId  Integer  not null references Choice,     -- The canonical choice
  check (ChoiceId <> AliasId)                 -- Circular aliases are no good
);


-- unnamed association
create table BallotStyleContestAssociation (
  StyleId    Integer  references BallotStyle,
  ContestId  Integer  references Contest,
  primary key (StyleId, ContestId)
);


-- unnamed association
create table BallotStyleReportingContextAssociation (
  StyleId           Integer  references BallotStyle,
  ReportingContext  Text     references ReportingContext,
  primary key (StyleId, ReportingContext)
);


-- unnamed association
create table BallotReportingContextAssociation (
  BallotId          Integer  references Ballot,
  ReportingContext  Text     references ReportingContext,
  primary key (BallotId, ReportingContext)
);
```

3.2 Conveniences

The view ReportingContextAssociationMerge merges reporting contexts inherited from the ballot style with reporting contexts specified on ballot instances. Duplicates are suppressed.

```
create view ReportingContextAssociationMerge (BallotId, ReportingContext) as
    select BallotId, ReportingContext
      from BallotReportingContextAssociation
  union
    select BallotId, ReportingContext
      from Ballot natural join BallotStyleReportingContextAssociation;
```

The view VotableChoices identifies all canonical Choices for which a valid VoterInput could exist (those contained in the applicable BallotStyles), excluding aliases.

```
create view VotableChoices (BallotId, ChoiceId) as
  select BallotId, ChoiceId
    from Ballot
      natural join BallotStyleContestAssociation
      natural join Choice
    where ChoiceId not in
      (select AliasId from Alias);
```

The view ReportingContextContestAssociation identifies all Contests that are relevant in a given ReportingContext. This includes those appearing in a BallotStyle associated with the context and those appearing in a Ballot associated with the context. A BallotStyle association can make a Contest relevant even if there are no applicable Ballots.

```
create view ReportingContextContestAssociation (ReportingContext, ContestId) as
    select ReportingContext, ContestId
      from BallotStyleReportingContextAssociation
        natural join BallotStyleContestAssociation
  union
    select ReportingContext, ContestId
      from BallotReportingContextAssociation
        natural join Ballot
        natural join BallotStyleContestAssociation;
```

The view FilteredContextContestAssociation is the same as ReportingContextContestAssociation except it excludes ranked order contests.

```
create view FilteredContextContestAssociation (ReportingContext, ContestId) as
  select ReportingContext, ContestId
    from ReportingContextContestAssociation
      natural join Contest
    where CountingLogic <> 'Ranked order';
```

The view FilteredContextChoiceAssociation identifies all Choices that are relevant in a given ReportingContext, excluding aliases and choices from ranked order contests.

```
create view FilteredContextChoiceAssociation (ReportingContext, ChoiceId) as
  select ReportingContext, ChoiceId
    from FilteredContextContestAssociation
      natural join Choice
    where ChoiceId not in
      (select AliasId from Alias);
```

The views BallotCounts, BallotCountsByConfiguration, BallotCountsByCategory, BallotCountsByCategoryAndConfiguration, BlankBallotCounts, and BlankBallotCountsByConfiguration produce the ballot counts that are required in post-voting reports.

BallotCounts and BlankBallotCounts report zeroes for contexts having no applicable ballots. The other views suppress rows pertaining to combinations of context, category and configuration that have no applicable ballots.

```
create view BallotCounts (ReportingContext, Read, Counted) as
  select Name, count(BallotId), count (nullif (Accepted, false))
    from Ballot
      natural join ReportingContextAssociationMerge
      right outer join ReportingContext on (Name = ReportingContext)
    group by Name;

create view BallotCountsByConfiguration (ReportingContext, StyleId,
                                         Read, Counted) as
  select ReportingContext, StyleId, count(*), count (nullif (Accepted, false))
    from Ballot natural join ReportingContextAssociationMerge
    group by ReportingContext, StyleId;

create view BallotCountsByCategory (ReportingContext, Category,
                                    Read, Counted) as
  select ReportingContext, Category, count(*), count (nullif (Accepted, false))
    from Ballot
      natural join ReportingContextAssociationMerge
      natural join BallotCategoryAssociation
    group by ReportingContext, Category;

create view BallotCountsByCategoryAndConfiguration (ReportingContext, StyleId,
                                                    Category, Read, Counted) as
  select ReportingContext, StyleId, Category, count(*),
         count (nullif (Accepted, false))
    from Ballot
      natural join ReportingContextAssociationMerge
      natural join BallotCategoryAssociation
    group by ReportingContext, StyleId, Category;

create view BlankBallot (BallotId, StyleId, Accepted) as
  select BallotId, StyleId, Accepted
    from Ballot
    where BallotId not in
      (select BallotId from VoterInput);

create view BlankBallotCounts (ReportingContext, Read, Counted) as
  select Name, count(BallotId), count (nullif (Accepted, false))
    from BlankBallot
      natural join ReportingContextAssociationMerge
      right outer join ReportingContext on (Name = ReportingContext)
    group by Name;

create view BlankBallotCountsByConfiguration (ReportingContext, StyleId,
                                              Read, Counted) as
  select ReportingContext, StyleId, count(*), count (nullif (Accepted, false))
    from BlankBallot natural join ReportingContextAssociationMerge
    group by ReportingContext, StyleId;
```

3.3 Adaptation

Converting the raw voter inputs into the effective voter inputs required by the logic model involves alias reconciliation, implementation of straight party voting, and generation of default (0) values for ballot positions that were not voted.

The VoterInput table has a primary key on (BallotId, ChoiceId), so there is at most one row for any given ballot position on any given ballot. Deliberately, the adaptation views do not preserve this constraint in the event that double votes result from alias reconciliation or straight party voting. Both of these cases are treated as errors for testing purposes, and the errors are most easily located by looking for duplicate keys. This is done by the integrity view DoubleVotes (see Section 3.4).

AntiAliasedVoterInput provides a view of VoterInput in which all choices have been "canonicalized."

```
create view AntiAliasedVoterInput (BallotId, ChoiceId, Value) as
  select BallotId, coalesce (Alias.ChoiceId, VoterInput.ChoiceId), Value
    from VoterInput left outer join Alias
      on VoterInput.ChoiceId = Alias.AliasId;
```

VoterInputMerge provides a view over AntiAliasedVoterInput in which the side-effects implied by straight party votes have been incorporated. If a straight party contest is overvoted, it has no side-effects. (ImpliedStraightPartyVotes is defined below.)

```
create view VoterInputMerge (BallotId, ChoiceId, Value) as
    select BallotId, ChoiceId, Value from AntiAliasedVoterInput
  union all
    select BallotId, ChoiceId, Value from ImpliedStraightPartyVotes;
```

Finally, the view EffectiveInput generates zeroes for ballot positions that were not voted.

```
create view EffectiveInput (BallotId, ChoiceId, Value) as
  select BallotId, ChoiceId, coalesce (Value, 0)
    from VotableChoices natural left outer join VoterInputMerge;
```

The definition of ImpliedStraightPartyVotes is provided here for completeness, though there is no reason that it would ever be used anywhere except in VoterInputMerge.

```
-- Straight party contests are implicitly 1-of-M contests.  If a
-- straight party contest is overvoted, it is irrelevant.

create view ValidStraightPartyVotes (BallotId, Party) as
  select BallotId, max(Name)      -- There can be only one.  See below.
    from AntiAliasedVoterInput
      natural join Choice
      natural join Contest
    where CountingLogic = 'Straight party selection'
    group by BallotId
    having sum(Value) = 1;        -- There can be only one.

-- Generate the implied straight party votes only for contests that
-- appear in the ballot style.
```

Constraint	Integrity view(s)
Constraint I	OutOfRangeVoterInputs, OutOfRangeEndorsements
Constraint II	N/A, enforced by SQL check constraint
Constraint III	N/A, enforced by SQL check constraint
Constraint IV	UnreportedBallots
Constraint V	ExtraneousInputs
Constraint VI	MoreThanOneStraightPartyContest
Constraint VII	CircularStraightPartyEndorsements
Constraint VIII	NonExistentParties
Constraint IX	N/A, enforced by SQL primary key constraint
Constraint X	DoubleVotes
Constraint XI	ScratchVotes
Constraint XII	DoubleIndirectAliases
Constraint XIII	CrossContestAliases
Constraint XIV	EndorsedAliases
Constraint XV	TooManyWriteIns

Table 1: Integrity checks

```
create view ImpliedStraightPartyVotes (BallotId, ChoiceId, Value) as
  select BallotId, ChoiceId, Value
    from VotableChoices
      natural join Endorsement
      natural join ValidStraightPartyVotes;
```

3.4 Integrity checks

For those integrity constraints that are too complex to code directly as SQL constraints within the tables, a series of views exists to look for problems. All of the integrity checking views should always be empty. If data appear in any of the views, the input was invalid and the results of the model will be invalid. The integrity views are listed in Table 1 but their definitions have been elided.

3.5 Translation of logic model

The following transforms are used to render the logic model as SQL.

1. Each function is replaced by a view in which the parameters form the primary key and the last column is the value of the function.

2. Time parameters (t) are factored out. All views implicitly project results for the time t corresponding to the current state of the database.

3. When a function takes both a contest and a choice as parameters, the contest parameter is omitted. With the data model used here, the Contest can be inferred from the Choice.

4. Logic is translated into those SQL constructs that are most transparently equivalent.
5. Ranked order contests, which are not handled by the logic model, are suppressed.
6. Irrelevant values, such as zero tallies for choices that do not appear in the applicable ballot style or contests that are not relevant in the applicable reporting context, are suppressed.

The following subsections first quote relevant portions of the logic model, then describe their analogs in the schema. Some terms from the logic model are elided from this discussion. For complete information on the logic model, please refer to the VVSG [1].

3.5.1 $S(c, r, t, v)$

> Ballot v's vote with respect to candidate or choice c in contest r as of time t. For checkboxes and the like, the value is 1 (selected) or 0 (not selected). For cumulative voting, the value is the number of votes that v gives to candidate or choice c in contest r. If the applicable ballot style does not include contest r, $S(c, r, t, v) = 0$.

The quaternary function S is implemented by the view EffectiveInput defined in Section 3.3. The current value of $S(c, r, t, v)$ is obtained by selecting Value where BallotId $= v$ and ChoiceId $= c$.

3.5.2 $S(r, t, v)$

> The total number of votes that ballot v has in contest r as of time t.

$$S(r, t, v) = \sum_{c \in C(r,t)} S(c, r, t, v)$$

The ternary function S is implemented by the view S. The current value of $S(r, t, v)$ is obtained by selecting S_val where ContestId $= r$ and BallotId $= v$. The view S contains rows only for contests that actually appear on the ballot according to its ballot style. All others are defined to be 0.

```
create view S (ContestId, BallotId, S_val) as
  select ContestId, BallotId, sum(Value)
    from EffectiveInput
      natural join Choice
      natural join Contest
    where CountingLogic <> 'Ranked order'
    group by ContestId, BallotId;
```

VotesByContestAndContext is a convenience to retrieve all of the S_val vote counts for each relevant combination of context and contest. For each relevant combination of context and contest that contains no ballots, there is a single row with nulls in the last three columns.

```
create view VotesByContestAndContext (ContestId, N, ReportingContext,
                                      BallotId, Accepted, S_val) as
  select ContestId, N, ReportingContext, BallotId, Accepted, S_val
    from FilteredContextContestAssociation
      natural join Contest
      natural left outer join (S natural join ReportingContextAssociationMerge)
      natural left outer join Ballot;
```

3.5.3 $S'(c, r, t, v)$

Ballot v's vote with respect to candidate or choice c in contest r as accepted for counting purposes (i.e., valid votes only), as of time t.

$$t \geq t_E \rightarrow S'(c, r, t, v) = \begin{cases} S(c, r, D(v), v) & \text{if } S(r, D(v), v) \leq N(r) \wedge A(t, v) \\ 0 & \text{otherwise} \end{cases}$$

The quaternary function S' is implemented by the view SPrime. The current value of $S'(c, r, t, v)$ is obtained by selecting SPrime_val where ChoiceId = c and BallotId = v.

```
create view SPrime (ChoiceId, BallotId, SPrime_val) as
  select ChoiceId, BallotId,
    case
      when S_val <= N and Accepted then Value
      else 0
    end
    from EffectiveInput
      natural join Choice
      natural join Contest
      natural join Ballot
      natural join S;
```

3.5.4 $T(c, j, r, t)$

The vote total for candidate or choice c in contest r and reporting context j as of time t. This does not include votes that are invalid due to overvoting or votes from ballots for which $A(t, v)$ is false.

$$t \geq t_E \rightarrow T(c, j, r, t) = \sum_{v \in V(j, t_E)} S'(c, r, t_E, v)$$

The quaternary function T is implemented by the view T. The current value of $T(c, j, r, t)$ is obtained by selecting T_val where ChoiceId = c and ReportingContext = j.

```
create view T (ChoiceId, ReportingContext, T_val) as
  select ChoiceId, ReportingContext, coalesce (sum (SPrime_val), 0)
    from FilteredContextChoiceAssociation
      natural left outer join
        (SPrime natural join ReportingContextAssociationMerge)
    group by ChoiceId, ReportingContext;
```

TSum is a convenience that sums T_val by contest.

```
create view TSum (ContestId, ReportingContext, TSum_val) as
  select ContestId, ReportingContext, sum(T_val)
    from T natural join Choice
    group by ContestId, ReportingContext;
```

3.5.5 $O(j, r, t)$

For a given contest and reporting context, the number of overvotes in read ballots for which $A(t, v)$ is true as of time t. Each ballot in which contest r is overvoted contributes $N(r)$ to $O(j, r, t)$.

$$t \geq t_E \rightarrow O(j, r, t) = \sum_{v \in V(j, t_E)} \begin{cases} N(r) & \text{if } S(r, D(v), v) > N(r) \wedge A(t, v) \\ 0 & \text{otherwise} \end{cases}$$

The ternary function O is implemented by the view O. The current value of $O(j, r, t)$ is obtained by selecting O_val where ContestId $= r$ and ReportingContext $= j$.

```
create view O (ContestId, ReportingContext, O_val) as
  select ContestId, ReportingContext, coalesce (sum (
      case
        when S_val > N and Accepted then N
        else 0
      end                                              ), 0)
    from VotesByContestAndContext
    group by ContestId, ReportingContext;
```

3.5.6 $U(j, r, t)$

For a given contest and reporting context, the number of undervotes in read ballots for which $A(t, v)$ is true as of time t. A given ballot contributes at most $N(r)$ to $U(j, r, t)$. Ballot styles that do not include contest r do not contribute to this total.

$$t \geq t_E \rightarrow U(j, r, t) = \sum_{v \in V(j, t_E)} \begin{cases} N(r) - S(r, D(v), v) & \text{if } S(r, D(v), v) \leq N(r) \wedge A(t, v) \\ 0 & \text{otherwise} \end{cases}$$

The ternary function U is implemented by the view U. The current value of $U(j, r, t)$ is obtained by selecting U_val where ContestId $= r$ and ReportingContext $= j$.

```
create view U (ContestId, ReportingContext, U_val) as
  select ContestId, ReportingContext, coalesce (sum (
      case
        when S_val <= N and Accepted then N - S_val
        else 0
      end                                              ), 0)
    from VotesByContestAndContext
    group by ContestId, ReportingContext;
```

3.5.7 $K(j, r, t)$

For a given contest and reporting context, the number of read ballots for which $A(t, v)$ is true as of time t (i.e., the number of ballots that should be counted). Ballot styles that do not include contest r do not contribute to this total.

The ternary function K is implemented by the view K. The current value of $K(j, r, t)$ is obtained by selecting K_val where ContestId $= r$ and ReportingContext $= j$.

```
create view K (ContestId, ReportingContext, K_val) as
  select ContestId, ReportingContext,
    (select count(*)
       from Ballot
         natural join ReportingContextAssociationMerge
         natural join BallotStyleContestAssociation
       where ReportingContextAssociationMerge.ReportingContext
         = FilteredContextContestAssociation.ReportingContext
       and BallotStyleContestAssociation.ContestId
         = FilteredContextContestAssociation.ContestId
       and Accepted)
    from FilteredContextContestAssociation;
```

3.5.8 Balance

Every vote must be accounted for.

$$t \geq t_E \rightarrow \sum_{c \in C(r,t)} T(c, j, r, t) + O(j, r, t) + U(j, r, t) = K(j, r, t) \times N(r)$$

A check for this assertion is implemented by the view Balance. The current difference between $\sum_{c \in C(r,t)} T(c, j, r, t) + O(j, r, t) + U(j, r, t)$ and $K(j, r, t) \times N(r)$ is obtained by selecting Discrepancy where ContestId $= r$ and ReportingContext $= j$. Discrepancy should always be zero.

```
create view Balance (ContestId, ReportingContext, Discrepancy) as
  select ContestId, ReportingContext, K_val * N - (TSum_val + O_val + U_val)
    from K
      natural join TSum
      natural join O
      natural join U
      natural join Contest;
```

4 Test Suite

The test suite was tested with PostgreSQL 8.2.3 [4] running on a GNU/Linux operating system. It uses extensions to the SQL standard [2] that might not function as intended with other databases. The report generator was tested with g++ 4.1.2 [5] and Boost 1.33.1 [6].

Specific software is identified in this paper to foster understanding. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the software identified is necessarily the best available for the purpose.

File	Purpose
Infrastructure-IntegrityChecks.sql	Show contents of all integrity views.
Infrastructure-Report	Generate post-voting report for a specified ReportingContext; calculate the report total volume.
Infrastructure-TestFooter.sql	Print "END TEST CASE OUTPUT" footer.
Infrastructure-TestHeader.sql	Print "BEGIN TEST CASE OUTPUT" and timestamp and configure verbosity of output for all test cases.
Infrastructure-VoteSchema.sql	Create the schema.
runtest	Shell script to execute a test case.

Table 2: Test suite infrastructure

4.1 Installation

The report generator, Infrastructure-Report, is the only thing in the test suite that needs to be compiled. It is packaged with GNU automake [7], so all usual GNU tricks should work. Help on configuration options can be found in the INSTALL file or obtained by entering `./configure --help`.

Normally, one should only need to do the following to compile Infrastructure-Report.

```
bash-3.1$ ./configure
bash-3.1$ make
```

However, in the event that PostgreSQL and/or Boost are installed in nonstandard locations, an invocation such as the following might be required.

```
bash-3.1$ ./configure \
> CPPFLAGS="-I/usr/local/pgsql/include -I/usr/local/include/boost-1_33_1" \
> LDFLAGS="-L/usr/local/pgsql/lib"
bash-3.1$ make
```

Use of the Boost C++ libraries is optional. If they are not provided, the only consequence is a static (compile-time) assertion is changed to a run-time assertion.

4.2 Infrastructure

Infrastructure files are listed in Table 2. Only the runtest script need be invoked by the user. All other infrastructure files are invoked as needed by the test cases.

4.2.1 runtest

A test case is executed by changing the current working directory to the directory containing the test suite and invoking the runtest script with the file name of the test case as the first parameter. The runtest script resets the database to an initial state and then feeds the test case to the SQL interpreter. No database named votetest other than the one created by the test suite should exist or it will be destroyed.

Bit	Meaning
00001	Incorrect usage
00010	No such reporting context
00100	Exception on attempt to connect to database
01000	Exception while connected
10000	Exception on attempt to disconnect from database

Table 3: Infrastructure-Report return codes

```
#!/bin/bash
if test -z "$1"; then
  echo Usage: runtest test-file-name.sql
else
  dropdb votetest
  createdb votetest
  psql votetest < $1
fi
```

4.2.2 Infrastructure-Report

Usage: `Infrastructure-Report context-name`. A no-frills, plain-ASCII post-voting report for the specified ReportingContext is sent to standard output. As a convenience to test labs, the report total volume needed for the accuracy test protocol of the VVSG is also calculated and reported. A sample report is shown in Figure 2.

Infrastructure-Report is normally invoked by individual test cases and need not be used directly. If it is invoked from a shell script, the codes that it returns to the shell are listed in Table 3. A return of 0 indicates success; other values indicate one or more problems as encoded by individual bits. Consult the standard error output of the program for additional details on the failure or failures that occurred.

If an error similar to the following occurs when Infrastructure-Report is invoked:

```
./Infrastructure-Report: error while loading shared libraries:
  libecpg.so.5: cannot open shared object file: No such file or directory
```

The solution is to add a command like the following to ∼/.bash_profile or another script that is always executed, specifying the location of the library that was not found.

```
export LD_LIBRARY_PATH=/usr/local/pgsql/lib
```

4.3 Level 0 (test suite self-tests)

4.3.1 Baseline

The test case 0-integrity-Baseline.sql verifies that the integrity views show no false positives on the base state for integrity tests.

Report for context Precinct 1 generated 2007-03-21 09:19-0400

BALLOT COUNTS

Configuration		Read	Counted
Total		13	13
	Blank	1	1
Precinct 1 Style		13	13
	Blank	1	1

VOTE TOTALS

Straight party, vote for at most 1	
Bipartisan Party	1
Moderate Party	1
Overvotes	1
Undervotes	10
Counted ballots	13
Balance	0
President, vote for at most 1	
Car Tay Fower	4
Tayra Tree	3
Beeso Tu (Moderate Party)	2
Oona Won (Bipartisan Party)	1
Nada Zayro	0
Overvotes	1
Undervotes	2
Counted ballots	13
Balance	0

Report total volume: 108
- Includes optional reporting of blank ballots.
- Excludes separate reporting of ballots cast vs. read.

Figure 2: Sample report

Constraint	Test case(s)
Constraint I	0-integrity-OutOfRangeInput.sql, 0-integrity-OutOfRangeEndorsement.sql
Constraint II	N/A, enforced by SQL check constraint
Constraint III	N/A, enforced by SQL check constraint
Constraint IV	0-integrity-UnreportedBallots.sql
Constraint V	0-integrity-ExtraneousInput.sql
Constraint VI	0-integrity-MoreThanOneStraightPartyContest.sql
Constraint VII	0-integrity-CircularEndorsement.sql
Constraint VIII	0-integrity-NonExistentParties.sql
Constraint IX	N/A, enforced by SQL primary key constraint
Constraint X	0-integrity-AliasDoubleVotes.sql
Constraint XI	0-integrity-ScratchVotes.sql
Constraint XII	0-integrity-DoubleIndirectAlias.sql
Constraint XIII	0-integrity-CrossContestAliases.sql
Constraint XIV	0-integrity-EndorsedAlias.sql
Constraint XV	0-integrity-TooManyWriteIns.sql

Table 4: Constraint violation tests

4.3.2 Constraint violations

The operation of schema constructs designed to detect violations of the constraints specified in Section 2.5 is verified by test cases that deliberately violate them. The test cases are listed in Table 4.

4.4 Level 1 (trivial tests)

Level 1 trivial tests are basic sanity checks that are not intended to challenge the abilities of any implementation. The reason for having them is that it is far easier to troubleshoot operational difficulties with a trivial test than with a realistically sized scenario. The test cases are listed in Table 5. Note that the documented assumption attached to test case 1-trivial-AbsenteeByCategory.sql means that it is not applicable to some *Absentee voting* systems.

Test case	Applies to	Description
1-trivial-1ofM.sql	*Voting system*	Trivial 1-of-M contest, no write-ins, no rejected ballots.
1-trivial-AbsenteeByCategory.sql	*Absentee voting*[1]	Trivial 1-of-M contest with absentee ballots via categories.
1-trivial-AbsenteeBySpecialPrecinct.sql	*Absentee voting*	Trivial 1-of-M contest with absentee ballots via a special precinct and ballot style.
1-trivial-CrossPartyEndorsement.sql	*Cross-party endorsement*	Trivial straight party + 1-of-M contest with cross-party endorsement.
1-trivial-Cumulative.sql	*Cumulative voting*	Trivial cumulative voting contest, no write-ins, no rejected ballots.
1-trivial-NofM.sql	*N of M voting*	Trivial 2-of-M contest, no write-ins, no rejected ballots.
1-trivial-Primary.sql	*Primary elections*	Trivial primary election, no write-ins, no rejected ballots.
1-trivial-Provisional.sql	*Provisional / challenged ballots*	Trivial 1-of-M contest with accepted and rejected provisionals.
1-trivial-SplitPrecinct.sql	*Split precincts*	Trivial 1-of-M contest with a split precinct.
1-trivial-StraightParty.sql	*Straight party voting*	Trivial straight party + 1-of-M contest, no write-ins, no rejected ballots.
1-trivial-WriteIns.sql	*Write-ins*	Trivial 1-of-M contest with write-ins, no aliasing.
1-trivial-WriteInsAliases.sql	*Write-ins*	Trivial 1-of-M contest with write-ins and aliases.

Table 5: Trivial tests

[1]Assumption: system supports categorization of ballots. This test is not applicable to systems that require the creation of distinct ballot styles or reporting contexts to implement absentee voting.

4.5 Test cases yet to be written

The initial testing strategy available under separate cover lays out a detailed hierarchy of test case development. The following are merely additional notes on special cases that also need testing at the appropriate time.

- Ranked order (was promised for FY07)
- No candidates
- No votes
- Combination cumulative voting + endorsements
- Combination ranked order + endorsements
- Randomly generated CVRs
- 25 out of 87 ("Vote for 25 out of 87 is not unheard of."—Michael Shamos)

References

[1] Election Assistance Commission. *Voluntary Voting System Guidelines*, 2007 edition. To appear, http://www.eac.gov/.

[2] Information technology—Database languages—SQL. ISO/IEC 9075, International Organization for Standardization, 2003. http://www.iso.org/.

[3] OMG Unified Modeling Language specification, version 1.5. Document formal/2003-03-01, Object Management Group, March 2003. http://www.omg.org/cgi-bin/doc?formal/2003-03-01.

[4] PostgreSQL version 8.2.3, February 2007. http://www.postgresql.org/.

[5] GNU Compiler Collection version 4.1.2, February 2007. http://gcc.gnu.org/.

[6] Boost C++ Libraries version 1.33.1, December 2005. http://www.boost.org/.

[7] GNU Automake version 1.9.6, July 2005. http://www.gnu.org/software/automake/automake.html.